Compilació separada

Josep Maria Ribó

Setembre de 2004

1 Programes clients de classes

Quan una aplicació client necessita utilitzar les operacions d'una determinada classe `C`, importa (*inclou*) la interfície d'aquella classe, que ha estat definida al fitxer de capcelera (e.g., `c.h`) corresponent a la classe `C`.

Això s'aconsegueix amb la directiva de precompilació `#include`. Per exemple:

```
#include "nomFitxer.h"
```

.

> La inclusió d'un fitxer `c.h` que conté la interfície d'una classe `C` dóna dret a les funcions contingudes al fitxer que fa aquella inclusió (e.g., `client.cpp`) a usar les operacions i atributs definides a la part pública de `C`.
> El fitxer `client.cpp` es diu que conté un programa client de la classe `C`.

Exemple 1.1 *En aquest exemple presentem el llistat del fitxer* `provacadena.cpp` *que utilitza la classe* `Cadena`*, la interfície de la qual ha estat definida en el fitxer* `cadena.h`*.*

```
#include "cadena.h"

int main()
{
 char car;
 Cadena c("pepet");

 car=c.posicio(2);     //char posicio(int) es una operacio
                       //definida a la classe Cadena
 ....
}
```

Compilació d'un fitxer client

L'única informació relativa a la classe `C` que es necessita per tal de compilar el fitxer `client.cpp` és la que aporta el fitxer amb la interfície `c.h` (recordem que el fitxer `c.h` defineix la part pública i la privada de la classe. La part pública conté la llista d'operacions que constitueixen la seva interfície).

Així doncs, podem compilar el fitxer `client.cpp` fent:

```
g++ -c client.cpp
```

El resultat d'aquesta comanda és un fitxer `client.o` que encara no és executable perquè conté crides a operacions, la implementació de les quals està continguda en un altre fitxer (e.g., `c.cpp`).

Les implementacions de les operacions de `C`, contingudes a `c.cpp` no són incorporades fins el procés de *link* (o *muntatge*) del programa complet que serà l'encarregat de generar un executable. Això s'explica amb més detall a la secció 3.

Exemple 1.2 *El fitxer* `provacadena.cpp` *de l'exemple anterior es pot compilar de la manera següent:*

```
g++ -c provacadena.cpp
```

Aquesta comanda genera un fitxer `provacadena.o` *que esperarà ser muntat amb* `cadena.o`.

1.1 Utilització dels include

Les directives `#include` es poden utilitzar de maneres diferents:

- `#include "nomFitxer.h"`

  Inclou el fitxer `nomFitxer.h` cercant-lo al directori de treball i, si no hi és, als directoris per defecte del compilador on s'emmagatzemen els fitxers de capcelera que poden ser inclosos en un programa.

- `#include <nomFitxer.h>`

  Inclou el fitxer `nomFitxer.h` cercant-lo als directoris per defecte del compilador on s'emmagatzemen els fitxers de capcelera que poden ser inclosos en un programa.

  Exemple:

  ```
  #include <string.h>
  ```

  Aquesta manera d'incloure biblioteques és ja obsoleta i no estàndar. S'ha d'anar substituint progressivament per la que segueix.

- `#include <nomBiblioteca>`

  Exemple: `#include <iostream>`

  C++ defineix algunes biblioteques estàndar, per exemple:

  - `<iostream>` Classes estàndar d'E/S
  - `<string>` Classe `string` de C++
  - `<map>` Classe `map`, per definir taules amb elements de tipus `T`
  - `<vector>` Classe `vector` per definir arrays unidimensionals d'elements de tipus `T`
  - `<cstring>` (biblioteca de strings de C), `<cmath>` (biblioteca que defineix funcions matemàtiques en C), `<cstdlib>` (biblioteca que defineix funcions estàndar en C), `<cstdio>` (E/S en C)...
    Aquestes biblioteques procedeixen de C i se'ls ha canviat el nom afegint-hi, al nom tradicional que tenien en C, una `c` a l'inici. En C s'incloien fent, per exemple, `#include<string.h>`, però ara s'inclouran com la resta de biblioteques de C++ (i.e., `#include<cstring>`)

  L'estàndar `C++` permet la definició d'espais de noms per tal d'evitar la repetició d'identificadors (de classes, de funcions, etc.). S'ha creat un espai de noms específic, anomenat `std`, que inclou les biblioteques estàndar de C++ i també les de C.

  La forma correcta d'incloure una d'aquestes biblioteques serà acompanyant l'include de la instrucció `using`:

  ```
  #include <iostream>

  using namespace std;
  ```

2 Compilació condicional

La inclusió de fitxers amb la directiva `#include` pot donar lloc a redefinicions d'interfícies de classes.

Per exemple, el fitxer `practica.cpp` inclou `text.h` (que defineix la classe `Text`) i `cadena.h` (que defineix la classe `Cadena`). Però a la seva vegada, el fitxer `text.h` també inclou el fitxer `cadena.h`. Això provoca la redefinició de la classe `Cadena` i el consegüent error de compilació.

C++ permet resoldre aquest tipus de situació mitjançant les directives de *compilació condicional*. D'aquestes directives nosaltres usarem `ifndef` i `endif`.

Exemple 2.1 *En aquest exemple presentem l'ús de les directives* `ifndef` *i* `endif` *al fitxer* `cadena.h`.

```
#ifndef CADENA_H
#define CADENA_H

....contingut del fitxer cadena.h

#endif
```

La interpretació de les directives de precompilació que apareixen al fitxer és la següent:

Si l'etiqueta `CADENA_H` **no** ha estat definida prèviament durant la precompilació, es defineix tot seguit (amb la directiva `define` i es tenen en consideració per a la compilació totes les instruccions C++ que conté el fitxer fins arribar al final de la directiva condicional `endif`.

Si, contràriament, l'etiqueta `CADENA_H` **sí** està definida (s'ha definit prèviament amb una directiva `define`), totes les instruccions C++ i directives de precompilació que hi pogués haver fins el final de la instrucció condicional (`endif`) s'ignoren (i, per tant, no es redefineix la classe `Cadena`. Dit d'una altra manera, un segon `include` de `cadena.h` queda sense efecte i no provoca redefinicions d'elements).

3 Procés de compilació

Il.lustrem el procés de compilació que cal seguir per crear un executable (anomenat `practica`) que depén d'un seguit de fitxers que es mostren a la figura 1.

En aquesta figura, cadascun dels fitxers `.h` defineix la interfície d'una classe (e.g., `text.h` defineix la interfície de la classe `Text`). La implementació de les operacions d'aquella classe es troba al fitxer amb el mateix nom i extensió `.cpp` (e.g., `text.cpp`). Finalment, el fitxer `practica.cpp` és el fitxer que conté el programa principal, que és client de la resta de classes.

Procés de compilació:

1. Compilació (per separat) de tots els fitxers `.cpp`. La compilació de cadascun d'aquests fitxers crea un fitxer objecte (`.o`).

   Com ja hem dit, la compilació d'un fitxer `f.cpp` no necessita la implementació de les operacions que `f.cpp` utilitza i que són codificades en algun altre fitxer. Només necessita la interfície (capcelera) de les mateixes (per això n'hi ha prou amb la inclusió del fitxer que conté aquelles capceleres).

   L'opció de compilació `-c` força a que no es faci muntatge. Només compilació.

   ```
   $g++ -c practica.cpp     -->genera practica.o

   $g++ -c tema.cpp          -->genera tema.o
   ```

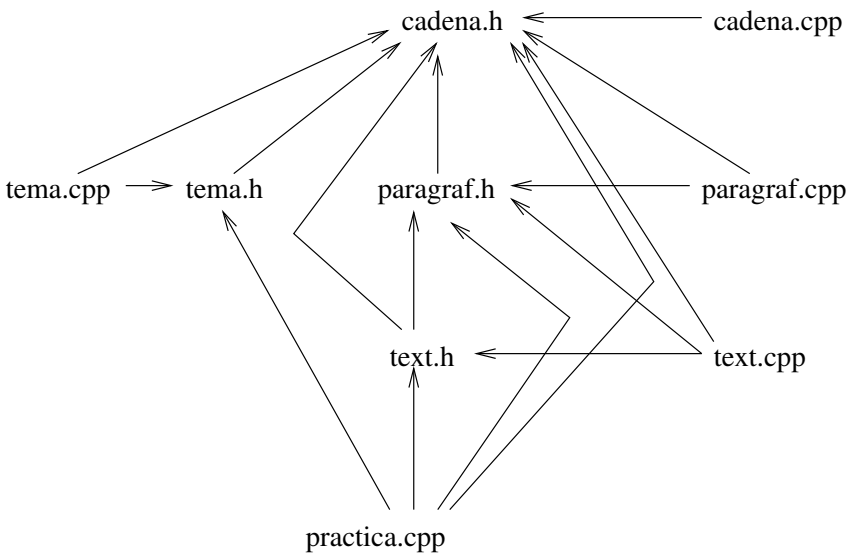

Figura 1: Diagrama de dependències.

```
$g++ -c cadena.cpp        -->genera cadena.o

$g++ -c text.cpp          -->genera text.o

$g++ -c paragraf.cpp      -->genera paragraf.o
```

2. Muntatge (*link*) de tots els fitxers objecte creats en el pas anterior per tal de generar un fitxer executable.

   Aquest muntatge reuneix les implementacions (ja compilades) de totes les operacions codificades en els diferents fitxers d'implementació.

   ```
   $g++ -o practica practica.o cadena.o tema.o paragraf.o text.o
   ```

   Aquesta instrucció genera un fitxer executable anomenat `practica`.

4 Automatització del procés de compilació

El procés de compilació separada d'un seguit de fitxers font pot ser complex o llarg si el nombre de fitxers implicats és elevat. Per simplificar-lo, existeix una eina (programa) anomenada *make* que permet automatitzar aquest procés de compilació separada.

El programa *make* es crida des de la línia de comandes i utilitza les instruccins de compilació que es descriuen en un fitxer anomenat *Makefile*. Seguidament describim com es pot dur a terme l'automatització del procés de compilació separada.

1. Crear un fitxer anomenat `Makefile` amb el següent contingut:

   ```
   practica: practica.o cadena.o paragraf.o text.o tema.o
   ```

```
	g++ -o practica practica3.o cadena.o tema.o paragraf.o text.o

practica.o: practica.cpp cadena.h tema.h paragraf.h text.h
	g++ -c practica.cpp

tema.o: tema.cpp cadena.h tema.h
	g++ -c tema.cpp

cadena.o: cadena.cpp cadena.h
	g++ -c cadena.cpp

text.o: text.cpp text.h paragraf.h cadena.h
	g++ -c text.cpp

paragraf.o: paragraf.cpp paragraf.h cadena.h
	g++ -c paragraf.cpp
```

El fitxer `Makefile` consta d'un o varis blocs amb el següent esquema global:

```
fitxer objectiu: fitxers dependencia
        accio per tal de generar el fitxer objectiu
```

Els fitxers dels quals depén cada fitxer font (.h o .cpp) apareixen a la figura 1.

2. El procés automàtic de compilació (o recompilació) s'engega fent:

   ```
   $make practica
   ```

   On *practica* és el nom del fitxer objectiu final del proés de compilació-muntatge (tal com apareix a la primera línia del fitxer `Makefile`).

Observacions:

- Al costat del fitxer objectiu s'escriuen els fitxers dels quals el fitxer objectiu depén. Això és, els fitxers tals que, si es modifiquen, obligaran a generar novament el fitxer objectiu.
- El programa `make` només genera aquells fitxers necessaris per construir el fitxer objectiu tals que no estiguin actualitzats (i.e. que la seva data de creació sigui anterior a la data de creació d'algun dels fitxers dels quals depenen).
- El fitxer `Makefile` ha de contenir les instruccions i les dependències per generar tots els fitxers involucrats en un procés de compilació-muntatge.
- Si el fitxer *objectiu final* coincideix amb el primer objectiu del fitxer `Makefile`, no cal escriure'l en la comanda.
- Es pot triar un altre nom pel fitxer `Makefile`. Si el nom escollit per aquest fitxer és `nouMake` la crida al programa make serà:

  ```
  $make -f nouMake
  ```

- Típicament, els objectius que apareixen a l'inici de cada bloc del fitxer `Makefile` són fitxers:

  ```
  fitxer objectiu: fitxers dependencia
          accio per tal de generar el fitxer objectiu
  ```

Però això no és obligatori. Vegem alguns exemples:

- **Exemple 1:**

```
totapractica: practica.o cadena.o paragraf.o text.o tema.o
     g++ -o practica practica3.o cadena.o tema.o paragraf.o text.o
```

  En aquest cas, l'objectiu es diu `totapractica` però el fitxer executable que acaba generant la instrucció és `practica`.
  Per generar el fitxer executable `practica` cal executar la comanda:

```
make totapractica
```

- **Exemple 2:**

```
netejar:
   rm -f *.o practica
```

  Aquí l'objectiu és `netejar`. Aquest objectiu no es correspon amb el nom de cap fitxer ni tampoc té cap dependència. Es crida: `make netejar` i provoca l'esborrat de tots els fitxers `.o` del directori i també del fitxer executable `practica`.

- El fitxer `Makefile` pot contenir també algunes definicions per tal de seleccionar adequadament les opcions de compilació. Tot seguit en mostrem un exemple:

```
CC=g++
CPPFLAGS= -Wall -g


practica: practica.o cadena.o paragraf.o text.o tema.o
$(CC) $(CFLAGS) -o practica practica.o cadena.o tema.o paragraf.o text.o

practica.o: practica.cpp cadena.h tema.h paragraf.h text.h
$(CC) $(CPPFLAGS) $(CFLAGS) -c practica3.cpp

tema.o: tema.cpp cadena.h tema.h
$(CC) $(CPPFLAGS) $(CFLAGS) -c tema.cpp

cadena.o: cadena.cpp cadena.h
$(CC) $(CPPFLAGS) $(CFLAGS) -c cadena.cpp

text.o: text.cpp text.h paragraf.h cadena.h
$(CC) $(CPPFLAGS) $(CFLAGS) -c text.cpp

paragraf.o: paragraf.cpp paragraf.h cadena.h
$(CC) $(CPPFLAGS) $(CFLAGS) -c paragraf.cpp
```